3IF – Février 2007
R. Laurini

# DS de BD relationnelles de Février 2007
# CORRECTION
# (Les corrections sont données en italique)

**Remarques générales**

1 – Il faut toujours expliquer ce que l'on fait ; dans la correction ci-joint, les textes en italiques ne sont pas des explications pour les étudiants afin qu'ils comprennent la solution, mais des éléments de réflexion qui donnent être donnés au professeur afin qu'il puisse juger de vos capacités de résolution de problème. Par exemple, une requête SQL sans commentaires AVANT se voit systématiquement attribuer 50 % seulement de la note globale ; les commentaires AVANT servent à expliquer le cheminement de votre raisonnement. S'ils sont rédigés après la requête, ils sont bien évidemment inutiles, donc pas notés : on ne vous demande pas de rédiger un manuel d'utilisation, mais d'expliquer votre démarche de manière claire, car un ingénieur doit toujours expliquer et argumenter avant de faire. De plus, expliquer un raisonnement par un schéma est une alternative souvent intéressante.

2 – Attention à la présentation. Des copies mal rédigées (brouillons) ou pleines de fautes d'orthographes pourront entraîner une réduction de un ou deux points.

3 – Systématiquement je demande une question à rédiger en langue anglaise avec 2 points de bonus (si l'anglais est excellent).

=============================================

A – Dans la création d'une BD relationnelle, quel est le rôle exact des mots-clés suivants `FOREIGN KEY, UNIQUE, REFERENCES, PCTFREE` ? Est-il possible de les combiner avec le mot-clé `NULL` (2 points) ?

*Le mot-clé `FOREIGN KEY` s'applique lorsque l'on désire qu'un attribut soit clé étrangère et le mot-clé `REFERENCES` indique la clé primaire correspondante. Le mot-clé `UNIQUE` est utilisé lorsqu'on désire que toutes les valeurs soient différentes.*

*L'extrait de la grammaire suivant montre qu'il n'est pas possible de combiner les trois premiers mots-clés avec `NULL`.*

inline_constraint::=

CONSTRAINT constraint_name
NOT NULL
UNIQUE
PRIMARY KEY
references_clause
CHECK ( condition )
constraint_state

out_of_line_constraint::=

CONSTRAINT constraint_name
UNIQUE ( column , )
PRIMARY KEY ( column , )
FOREIGN KEY ( column , ) references_clause
CHECK ( condition )
constraint_state

references_clause::=

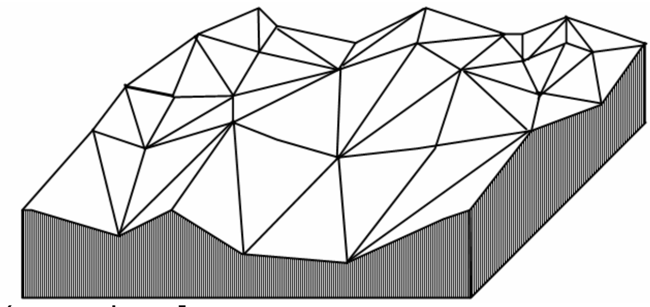
REFERENCES schema . object ( column )
ON DELETE CASCADE
SET NULL

*En ce qui concerne le dernier mot-clé `PCTFREE`, celui s'applique au niveau physique pour indiquer le pourcentage de place libre dans les blocs physiques en deçà duquel il ne sera plus possible d'insérer de nouvelles lignes. Puisque le mot-clé `PCTFREE` est totalement indépendant du mot-clé `NULL`, il n'y aucun problème à ce qu'ils soient combinés.*

B – Dans le cas d'un TRIGGER AFTER, détailler le mécanisme du fonctionnement lorsque le trigger doit annuler l'événement qui l'avait activé (2 points) ?

*Dans le cas d'un `TRIGGER AFTER`, l'ordre qui l'a activé a été exécuté auparavant, puis suivi du trigger en question. Si dans le corps du trigger on est amené à supprimer l'ordre qui l'avait lancé, grâce à l'appel d'une procédure du type `RAISE_APPLICATION_ERROR`, cela correspondra à un `ROLLBACK` qui fera que la base de données redeviendra dans l'état antérieur à l'ordre qui avait activé le `TRIGGER`.*

C – Soit la base de données suivante qui représente un modèle numérique de terrain basé sur des triangles : On possède un ensemble de points dont on a mesuré les coordonnées *x*, *y* et l'altitude *z*. Ces points sont regroupés en triangles dont ils forment les sommets basés sur leurs côtés appelés segments. Les relations de base sont les suivantes :



```
TRIANGLE   (no-triangle, no-segment1, no-segment2, no-segment3)
SEGMENT    (no-segment, no-sommet1, no-sommet2)
SOMMET     (no-sommet, x, y, z)
```

C1 – Pour chacune de ces trois relations, expliquer quelles sont les clés primaires et les clés étrangères. En déduire l'ordonnancement de la création des tables. Donner le code en SQL en tenant compte de TOUTES les contraintes d'intégrités. Donner également les `CREATE SEQUENCE`s (3 points).

*Dans cette formulation, il est clair que l'on a les éléments suivants*

| *Nom de la table* | *Clé primaire* | *Clés étrangères* |
|---|---|---|
| *TRIANGLE* | *no-triangle* | *no-segment1, no-segment2, no-segment3* |
| *SEGMENT* | *no-segment* | *no-sommet1, no-sommet2* |
| *SOMMET* | *no-sommet* | |

*Dès lors, il faudra créer les tables en premier SOMMET, puis SEGMENT et TRIANGLE en dernier.*

*Création de la table SOMMET : on pourra mettre la contrainte additionnelle spécifiant que les coordonnées doivent être valuées.*

```
CREATE SOMMET
( no-sommet integer primary key
, x number (10,5) NOT NULL
, y number (10,5) NOT NULL
);
```

*Création de la table SEGMENT : il semble important de spécifier que les deux extrémités du segment doivent être différentes ; ce qui donne :*

```
CREATE SEGMENT
(no-segment integer primary key
, no-sommet1  integer, foreign key references SOMMET(no-sommet)
, no-sommet2  integer, foreign key references SOMMET(no-sommet)
, CONSTRAINT CHECK no-sommet1 != no-sommet2
);
```

*Création de la table TRIANGLE : de même, il semble intéressant de spécifier que les trois segments doivent être différents :*

```
CREATE TRIANGLE
(no-triangle integer primary key
, no-segment1  integer, foreign key references SEGMENT(no-segment)
, no-segment2  integer, foreign key references SEGMENT(no-segment)
, no-segment3  integer, foreign key references SEGMENT(no-segment)
, CONSTRAINT CHECK no-segment1 != no-segment2,
, CONSTRAINT CHECK no-segment1 != no-segment3,
, CONSTRAINT CHECK no-segment2 != no-segment3,
) ;
```

*Pour la création des séquences, il suffira d'écrire :*

```
CREATE SEQUENCE seq-sommet START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq-segment START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq-striangle START WITH 1 INCREMENT BY 1;
```

C2 – Faire une vue donnant la liste des segments situés sur la bordure du terrain (3 points).

*Alors que les segments situés à l'intérieur ont tous deux triangles de chaque côté, les segments situés sur la bordure n'ont qu'un seul triangle (COUNT(*). Il existe plusieurs façons de rédiger une telle vue. En voici une en deux étapes, elle consistera en une première vue (TRIANGLE1) afin de normaliser la table triangle sur laquelle s'appuiera la vue BORDURE.*

```
CREATE VIEW TRIANGLE1 (no-triangle, no-segment AS
     SELECT   no-triangle, no-segment1 FROM TRIANGLE
     UNION
     (SELECT   no-triangle, no-segment2 FROM TRIANGLE
```

```
    UNION
    (SELECT   no-triangle, no-segment3 FROM TRIANGLE));

CREATE VIEW BORDURE (no-segment) AS
    SELECT no-segment FROM TRIANGLE1
    GROUP BY no-segment
    HAVING Count(*)=1;
```

C3 – Dans un tel modèle de terrain, si l'on veut connaître l'altitude d'un point quelconque du terrain, on se base sur l'approximation planaire suivante : $z=A{\times}x+B{\times}y+C$. Ces trois coefficients ($A$, $B$, $C$) ont été calculés et stockés dans la table suivante : `COEF (no-triangle, A, B, C)`. Rédiger la commande `ALTER` qui permet d'ajouter trois attributs (`A`, `B` et `C`) à la relation `TRIANGLE`. Quelles sont les deux façons pour charger les valeurs ? Donner le code de l'une d'entre elles (au choix). Supposons qu'il manque un ou plusieurs triangles dans la relation `COEF`, comment détecter (explications et code) cet oubli (3 points).

*Pour ajouter les trois colonnes à la table `TRIANGLE`, il suffit d'écrire*

```
ALTER TABLE TRIANGLE
    ADD A NUMBER (10,5)
    ADD C NUMBER (10,5)
    ADD D NUMBER (10,5);
```

*Pour charger les valeurs, il existe deux solutions, soit un `UPDATE`, soit un programme PL/SQL. Voici la solution `UPDATE` :*

```
UPDATE TRIANGLE
    SET A, B, C =   SELECT A, B, C
                    FROM COEF
                    WHERE COEF.no-triangle=TRIANGLE.no-triangle;
```

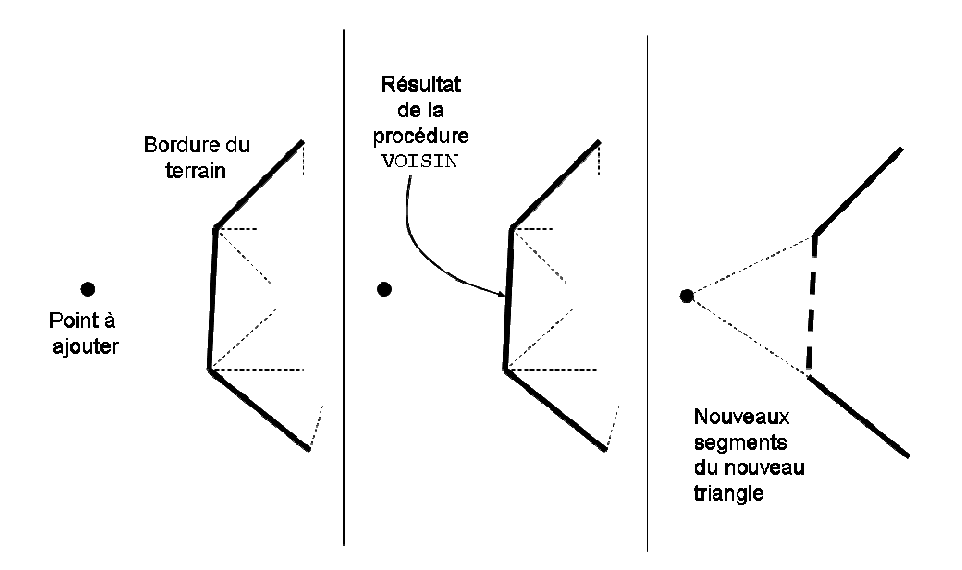
*Pour effectuer un contrôle de qualité, afin de connaître les divergences entre les deux tables, on peut faire des intersections :*

```
SELECT no-triangle from TRIANGLE MINUS (SELECT no-triangle FROM COEF);
SELECT no-triangle from COEF MINUS (SELECT no-triangle FROM TRIANGLE);
```

*Remarque : Comme dit dans l'énoncé de cet examen, bonus de deux points à ceux qui avaient répondu en bon anglais ou américain à cette question.*

C4 – On désire agrandir le terrain en ajoutant un nouveau point supposé extérieur au contour du terrain. Supposons que l'on dispose d'une procédure `VOISIN (no-sommet-ajoute, no-segment-voisin)` qui donne le numéro du segment de la bordure le plus proche du sommet que l'on désire ajouter. Après avoir donné les explications nécessaires, rédiger le trigger, qui à l'ajout d'un tel point, crée automatiquement les tuples nécessaires dans les tables `SEGMENT` et `TRIANGLE` (3 points).

*Le processus est donné dans le dessin suivant : suite à l'insertion d'un point, la procédure `VOISIN` nous donnera le segment le plus proche de la bordure. En s'appuyant sur le nouveau point à ajouter et ce segment, on créera deux nouveaux segments et un nouveau triangle.*




*Il s'agira d'un `TRIGGER AFTER INSERT`, c'est-à-dire que l'on effectuera d'abord l'ordre `INSERT` puis le trigger afin de respecter les contraintes d'intégrité. Les identificateurs des deux nouveaux segments et du nouveau triangle seront générés par l'appel aux `SEQUENCE`s. L'instruction pouvant lancer le trigger est de la forme :*

```
INSERT INTO TABLE SOMMET VALUES (nextval.seq-sommet, 345.56, 678.98, 45.89) ;
```

*Deux remarques pour le code du trigger :*

*1 – à cause des clés étrangères, il faudra d'abord créer les deux segments, puis le triangle ;*
*2 – comme nous avons deux clés de nouveaux identificateurs de segments, la clause `CURVAL` ne pourra pas être utilisée, d'où la nécessité de créer deux variables de no-segment.*

```
CREATE OR REPLACE TRIGGER
AFTER INSERT ON SOMMET
DECLARE
      no-segment-voisin integer ;
      no-segment-A integer ;
      no-segment-B integer ;
      no-point-A integer ;
      no-point-B integer ;
BEGIN
      VOISIN (:new.no-sommet, no-segment-voisin);
      SELECT no-point1, no-point2
            INTO no-point-A, no-point-B
            FROM SEGMENT
            WHERE no-segment = no-segment-voisin ;
      no-segment-A = nextval.seq-no-segment ;
      no-segment-B = nextval.seq-no-segment ;
      INSERT INTO TABLE SEGMENT (no-segment-A, no-point-A, :new.no-
      sommet) ;
      INSERT INTO TABLE SEGMENT (no-segment-B, no-point-B, :new.no-
      sommet) ;
      INSERT INTO TABLE TRIANGLE (nextval.seq-no-triangle, no-segment-
      voisin, no-segment-A, no-segment-B);
END ;
```

C5 – On modifie la table `SEGMENT` afin qu'elle devienne la suivante : `SEGMENT (no-segment, no-sommet1, no-sommet2, no-triangle1, no-triangle2)`, où `no-triangle1` et `no-triangle2` représentent les numéros des triangles situés de chaque côté du segment en question. Comment régler de manière simple les segments situés sur le bord du terrain qui n'ont qu'un seul voisin ? Après avoir

choisi une solution, préciser les conséquences en matière de contraintes d'intégrité et d'ordonnancement du chargement (4 points).

*A première vue, ajouter les deux attributs* `no-triangle1`, `no-triangle2` *revient à ajouter deux clés étrangères, mais c'est plus compliqué car cela reviendrait à avoir des clés étrangères avec références cycliques, ce qui est interdit. La seule solution est de créer des attributs normaux, et ensuite voir si l'on peut modifier la table avec un ordre du type* `ALTER TABLE ADD CONSTRAINT….`

*Une des solutions pour régler le problème des triangles de la bordure est de mettre la valeur* `NULL` *(au sens d'inapplicable) dans* `no-triangle2`*. Entre parenthèse, cette convention simplifierait l'écriture de la vue de la question C2.*

*Dans ces conditions, s'il on opte pour mettre des valeurs* `NULL` *dans* `no-triangle2`*, alors cet attribut ne pourra plus être clé étrangère (voir question A).*

*Dès lors, après la création de la* `TABLE SOMMET` *qui reste inchangée, on écrira pour la création de la nouvelle table SEGMENT (bien évidemment, les deux triangles sont différents) :*

```
CREATE SEGMENT
(no-segment integer primary key
, no-sommet1  integer, foreign key references SOMMET(no-sommet)
, no-sommet2  integer, foreign key references SOMMET(no-sommet)
, no-triangle1  integer NOT NULL
, no-triangle2  integer NULL
, CONSTRAINT CHECK no-sommet1 != no-sommet2
, CONSTRAINT CHECK no-segment1 != no-segment2
);
```

*Ensuite le CREATE TRIANGLE reste inchangé.*

*Puis tous les* `INSERT`*s pour le remplissage des tables, en commençant par remplir la table* `SOMMET`*, puis la table* `SEGMENT` *et enfin la table* `TRIANGLE`*.*

*Maintenant, suite à ce qui a été expliqué plus haut, on peut mettre la contrainte de clé étrangère sur* `no-triangle1`*. Ce qui donne :*

```
ALTER TABLE TRIANGLE
     ADD  CONSTRAINT  no-triangle1  FOREIGN  KEY  REFERENCES  TRIANGLE  (no-
triangle) ;
```

*Puisque l'on a une contrainte sur* `no-triangle2` *qui ressemble fortement à une clé étrangère, lors de l'adjonction d'un segment, on pourra rédiger un trigger qui se chargera de la vérification. En d'autres termes, puisque dans un tel cas de figure, on a affaire à une contrainte d'intégrité particulière qui ne peut être rédigée de manière déclarative, on résoudra le problème de manière procédurale.*